Alfred V. Aho
aho@cs.columbia.edu

The PLT Course at Columbia

CS
@CU

COMPUTER SCIENCE AT
COLUMBIA UNIVERSITY

Guest Lecture
PLT September 10, 2014

Outline

- **Course objectives**
- **Language issues**
- **Compiler issues**
- **Team issues**

Course Objectives

- Developing an appreciation for the critical role of software in today's world
- Discovering the principles underlying the design of modern programming languages
- Mastering the fundamentals of compilers
- Experiencing an in-depth capstone project combining language design and translator implementation

Plus Learning Three Vital Skills for Life

Project management

Teamwork

Communication both oral and written

The Importance of Software in Today's World

How much software does the world use today?

Guesstimate: around one trillion lines of source code

What is the sunk cost of the legacy software base?

$100 per line of finished, tested source code

How many bugs are there in the legacy base?

10 to 10,000 defects per million lines of source code

Alfred V. Aho
Software and the Future of Programming Languages
Science, v. 303, n. 5662, 27 February 2004, pp. 1331-1333

Why Take Programming Languages and Compilers?

To discover the marriage of theory and practice

To develop computational thinking skills

To exercise creativity

To reinforce robust software development practices

To sharpen your project management, teamwork and communication (both oral and written) skills

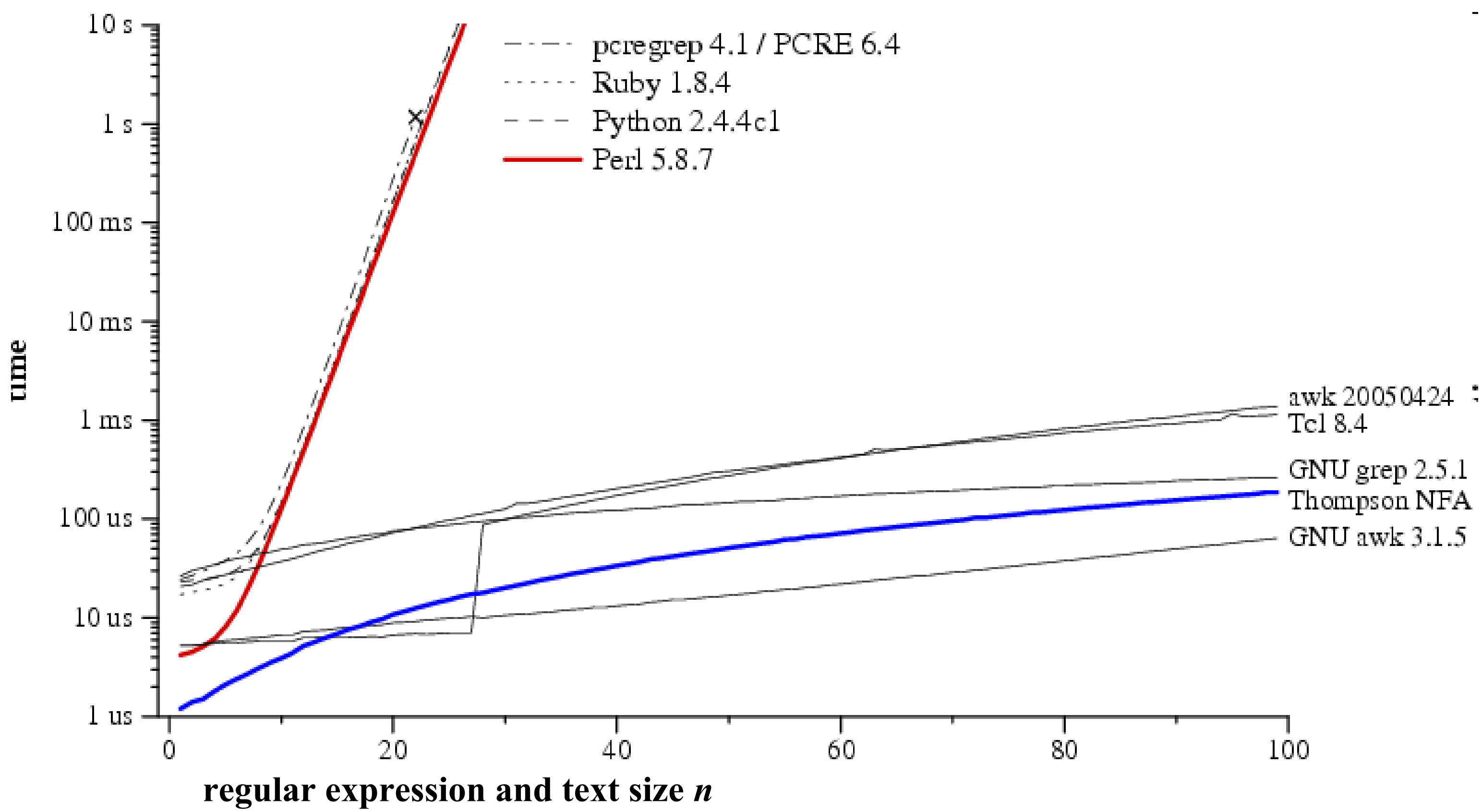
Why Take PLT?

To discover the beautiful marriage of theory and practice in compiler design

"Theory and practice are not mutually exclusive; they are intimately connected. They live together and support each other." [D. E. Knuth, 1989]

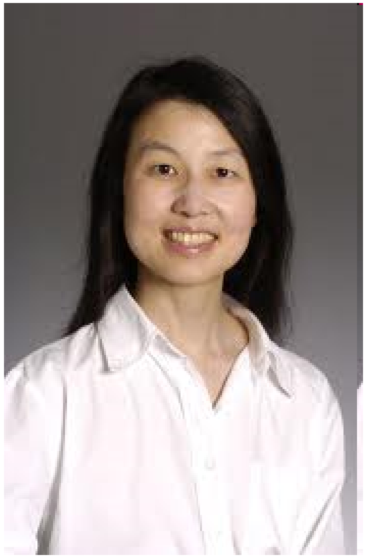
Theory in practice: regular expression pattern matching in Perl, Python, Ruby vs. AWK

Running time to check whether $a?^n a^n$ matches a^n



Russ Cox, *Regular expression matching can be simple and fast (but is slow in Java, Perl, PHP, Python, Ruby, ...)* [http://swtch.com/~rsc/regexp/regexp1.html, 2007]

Computational Thinking – Jeannette Wing

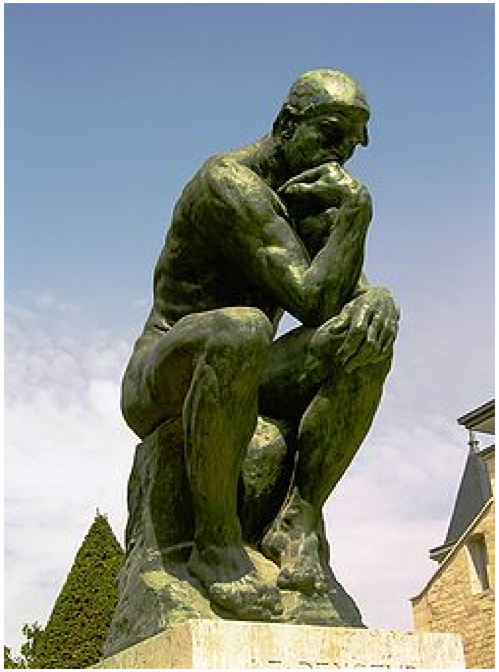


[Jeannette Wing, *Computational Thinking*, CACM, March, 2006]

Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability. Just as the printing press facilitated the spread of the three Rs, what is appropriately incestuous about this vision is that computing and computers facilitate the spread of computational thinking.

Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science. Computational thinking includes a range of mental tools that reflect the breadth of the field of computer science.

What is Computational Thinking?



The thought processes involved in formulating a problem and expressing its solution in a way that a computer – human or machine – can effectively carry it out

A. V. Aho
Computation and Computational Thinking
The Computer Journal 55:12, pp. 832-835, 2012

Jeannette M. Wing
Joe Traub 80th Birthday Symposium
Columbia University, November 9, 2012

What is a Programming Language?

A programming language is a notation for describing computations to people and to machines.

Evolutionary Forces Driving PL Changes

Increasing diversity of applications

Stress on increasing programmer productivity and shortening time to market

Need to improve software security, reliability and maintainability

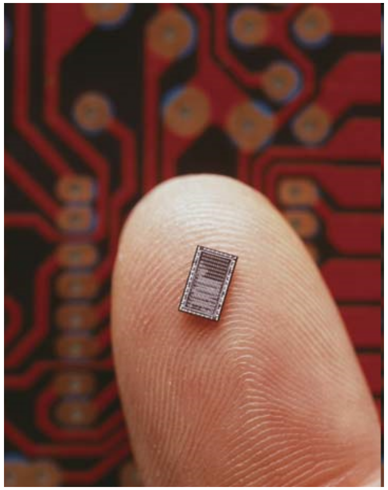
Emphasis on mobility and distribution

Support for parallelism and concurrency

New mechanisms for modularity and scalability

Trend toward multi-paradigm programming

Target Languages and Machines



Another programming language

CISCs

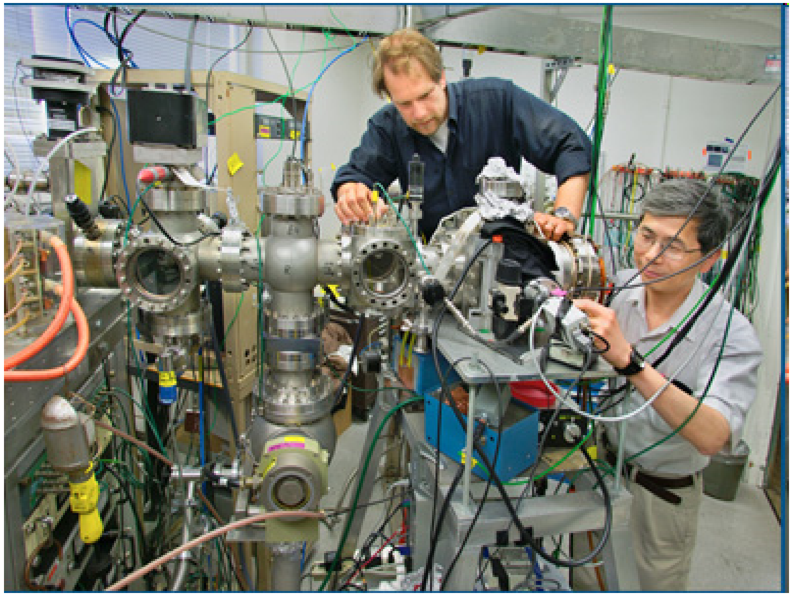
RISCs

Parallel machines

Multicores

GPUs

Quantum computers



How Many PLs are There Today?

Guesstimate: **thousands**

The website http://www.99-bottles-of-beer.net has programs in over 1,500 different programming languages and variations to print the lyrics to the song "99 Bottles of Beer."

"99 Bottles of Beer"

99 bottles of beer on the wall, 99 bottles of beer.
Take one down and pass it around, 98 bottles of beer on the wall.

98 bottles of beer on the wall, 98 bottles of beer.
Take one down and pass it around, 97 bottles of beer on the wall.

.

.

.

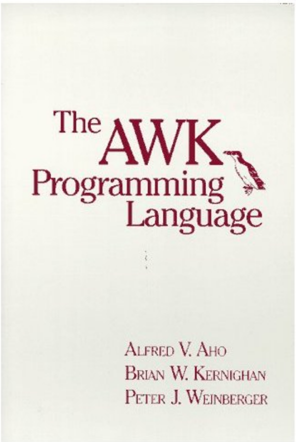
2 bottles of beer on the wall, 2 bottles of beer.
Take one down and pass it around, 1 bottle of beer on the wall.

1 bottle of beer on the wall, 1 bottle of beer.
Take one down and pass it around, no more bottles of beer on the wall.

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more, 99 bottles of beer on the wall.

[Traditional]

“99 Bottles of Beer” in AWK




```
BEGIN {
 for(i = 99; i >= 0; i--) {
  print ubottle(i), "on the wall,", lbottle(i) "."
  print action(i), lbottle(inext(i)), "on the wall."
  print
 }
}
function ubottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "No more", n - 1 ? "s" : "")
}
function lbottle(n) {
 return sprintf("%s bottle%s of beer", n ? n : "no more", n - 1 ? "s" : "")
}
function action(n) {
 return sprintf("%s", n ? "Take one down and pass it around," : \
                          "Go to the store and buy some more,")
}
function inext(n) {
 return n ? n - 1 : 99
}
```

"99 Bottles of Beer" in Perl

```
''=~(       '(?{'       .('`'       |'%')       .('['       ^'-')
.('`'       |'!')       .('`'       |',')       .'"'.       '\\$'
.'=='       .('['       ^'+')       .('`'       |'/')       .('['
^'+')       .'||'       .(';'       &'=')       .(';'       &'=')
.';-'       .'-'.       '\\$'       .'=;'       .('['       ^'(')
.('['       ^'.')       .('`'       |'"')       .('!'       ^'+')
.'_\\{'     .'(\\$'     .';=('.     '\\$=|'     ."\|".(     '`'^'.'
).(('`')|   '/').').'   .'\\"'.+(   '{'^'[').   ('`'|'"')   .('`'|'/'
).('['^'/') .('['^'/'). ('`'|',').( '`'|('%')). '\\".\\"'.( '['^('(')).
'\\"'.('['^ '#').'!!--' .'\\$=.\\"' .('{'^'['). ('`'|'/').( '`'|"\&").(
'{'^"\[").( '`'|"\"").( '`'|"\%").( '`'|"\%").( '['^(')')). '\\").\\"'.
('{'^'[').( '`'|"\/").( '`'|"\.").( '{'^"\[").( '['^"\/").( '`'|"\(").(
'`'|"\%").( '{'^"\[").( '['^"\,").( '`'|"\!").( '`'|"\,").( '`'|(',')).
'\\"\\}'.+( '['^"\+").( '['^"\)").( '`'|"\)").( '`'|"\.").( '['^('/')).
'+_,\\",'.( '{'^('[')). ('\\$;!').( '!'^"\+").( '{'^"\/").( '`'|"\!").(
'`'|"\+").( '`'|"\%").( '{'^"\[").( '`'|"\/").( '`'|"\.").( '`'|"\%").(
'{'^"\[").( '`'|"\$").( '`'|"\/").( '['^"\,").( '`'|('.')). ','.(('{')^
'[').("\["^ '+').("\`"| '!').("\["^ '(').("\["^ '(').("\{"^ '[').("\`"|
')').("\["^ '/').("\{"^ '[').("\`"| '!').("\["^ ')').("\`"| '/').("\["^
'.').("\`"| '.').("\`"| '$')."\,".( '!'^('+')). '\\",_,\\"' .'!'.("\!"^
'+').("\!"^ '+').'\\"'. ('['^',').( '`'|"\(").( '`'|"\)").( '`'|"\,").(
'`'|('%')). '++\\$="})' );$:=('.')^ '~';$~='@'| '(';$^=')'^ '[';$/='`';
```

“99 Bottles of Beer” in the Whitespace Language

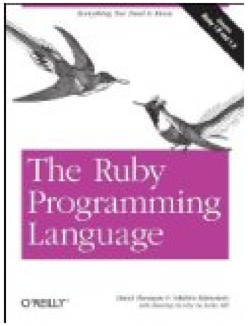
[Andrew Kemp, http://compsoc.dur.ac.uk/whitespace]

What are Today's Most Popular PLs?

tiobe.com	PyPL Index	RedMonk	StackOverflow
C	Java	Java/JavaScript	Java
Java	PHP		C#
Objective-C	Python	PHP	JavaScript
C++	C#	Python	PHP
Basic	C++	C#	Python
C#	C	C++/Ruby	C++
Python	Javascript		SQL
PHP	Objective-C	CSS	Objective-C
Perl	Ruby	C	C
JavaScript	Basic	Objective-C	Ruby
[www.tiobe.com, August 2014 Data from search engines]	[PyPL Index, August 2014 Tutorial searches on Google]	[redmonk.com, June 2014 Data from GitHub]	[langpop.corger.nl, August 2014 Data from GitHub]

Case Study 1: Ruby

- Ruby is a dynamic, OO scripting language designed by Yukihiro Matsumoto in Japan in the mid 1990s
- Characteristics: object oriented, dynamic, designed for the web, scripting, reflective
- Supports multiple programming paradigms including functional, object oriented, and imperative
- The three pillars of Ruby
  - everything is an object
  - every operation is a method call
  - all programming is metaprogramming
- Made popular by the web application framework Rails

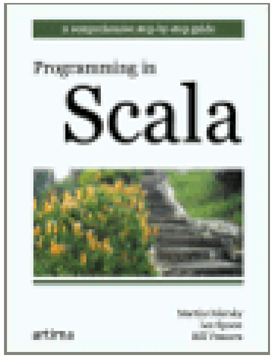


http://www.ruby-lang.org/en/about/

Case Study 2: Scala

- Scala is a multi-paradigm programming language designed by Martin Odersky at EPFL starting in 2001
- Characteristics: scalable, object oriented, functional, seamless Java interoperability, functions are objects, future-proof, fun
- Integrates functional, imperative and object-oriented programming in a statically typed language
- Functional constructs used for parallelism and distributed computing
- Generates Java byte code
- Used to implement Twitter
  - Katy Perry has 54 million followers
  - Barack Obama has 44 million followers

[http://twitaholic.com/]

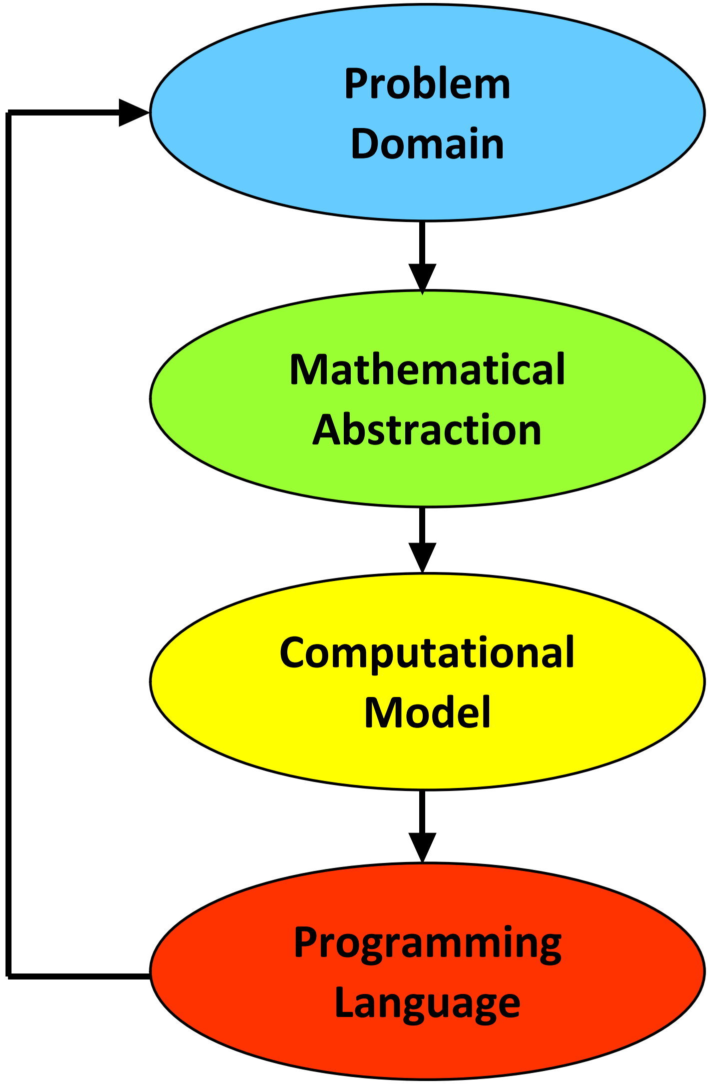


http://www.scala-lang.org/what-is-scala.html

Issues in Programming Language Design

- **Domain of application**
  - exploit domain restrictions for expressiveness, performance
- **Computational model**
  - simplicity, ease of expression
- **Abstraction mechanisms**
  - reuse, suggestivity
- **Type system**
  - reliability, security
- **Usability**
  - readability, writability, efficiency, learnability, scalability, portability

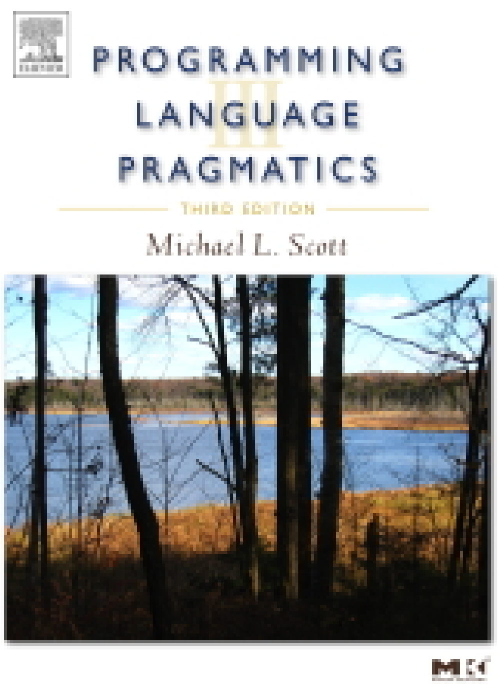
Computational Thinking in Language Design



Common Models of Computation in PLs

PLs are designed around a model of computation:

- **Procedural: Fortran (1957)**
- **Functional: Lisp (1958)**
- **Object oriented: Simula (1967)**
- **Logic: Prolog (1972)**
- **Relational algebra: SQL (1974)**



Computational Model Underlying AWK

AWK is a scripting language designed to perform routine data-processing tasks on strings and numbers

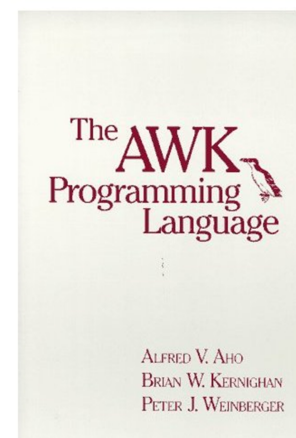
Use case: given a list of name-value pairs, print the total value associated with each name.

```
alice 10
eve 20
bob 15
alice 30
```

An AWK program is a sequence of pattern-action statements

```
    { total[$1] += $2 }
END { for (x in total) print x, total[x] }
```

```
eve 20
bob 15
alice 40
```



Kinds of Languages - I

- Declarative
  - Program specifies what computation is to be done
  - Examples: Haskell, ML, Prolog
- Domain specific
  - Many areas have special-purpose languages for creating applications
  - Examples: Lex for scanners, Yacc for parsers
- Functional
  - One whose computational model is based on lambda calculus
  - Examples: Haskell, ML

Kinds of Languages - II

- **Imperative**
  - Program specifies how a computation is to be done
  - Examples: C, C++, C#, Fortran, Java
- **Markup**
  - One designed for the presentation of text
  - Usually not Turing complete
  - Examples: HTML, XHTML, XML
- **Object oriented**
  - Program consists of interacting objects
  - Uses encapsulation, modularity, polymorphism, and inheritance
  - Examples: C++, C#, Java, OCaml, Smalltalk

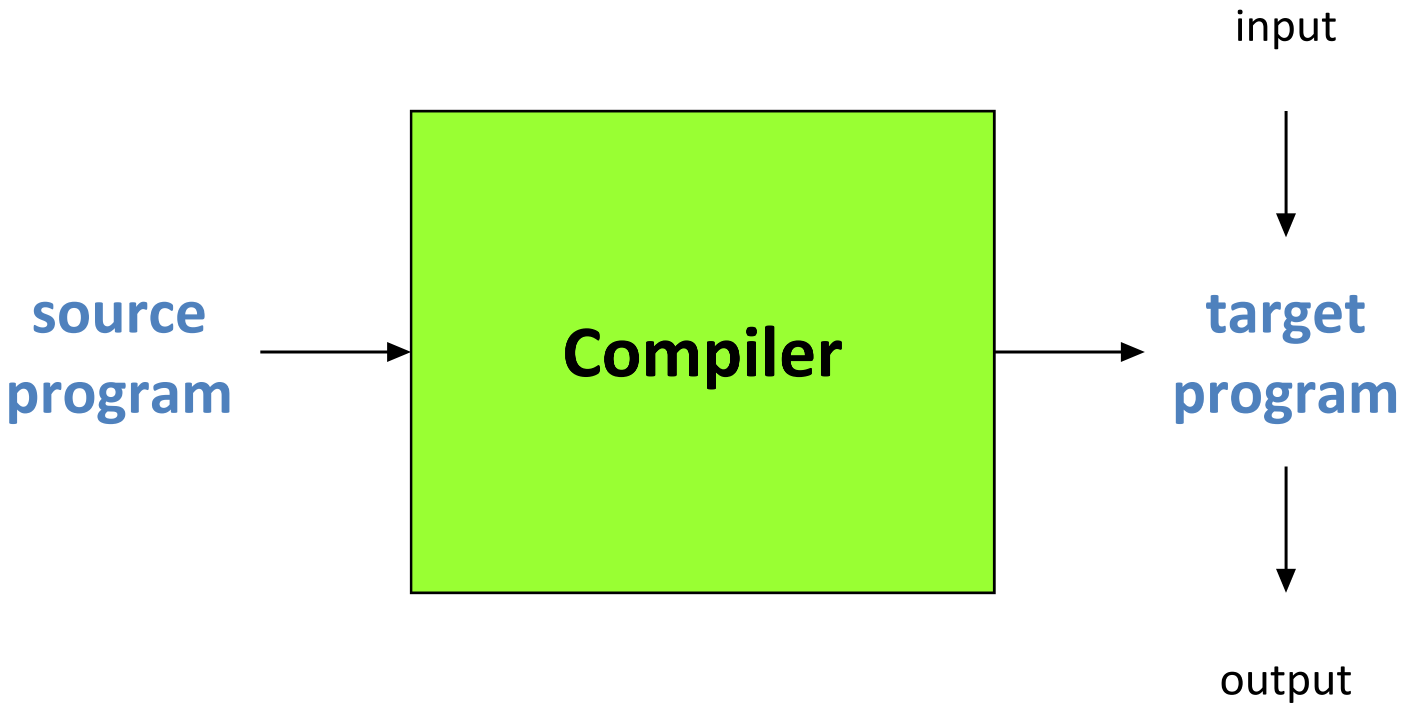
Kinds of Languages - III

- Parallel
  - One that allows a computation to run concurrently on multiple processors
  - Examples: CUDA, Cilk, MPI, POSIX threads, X10
- Scripting
  - An interpreted language with high-level operators for “gluing together” computations
  - Examples: Awk, Perl, PHP, Python, Ruby
- von Neumann
  - One whose computational model is based on the von Neumann architecture
  - Computation is done by modifying variables
  - Examples: C, C++. C#, Fortran, Java

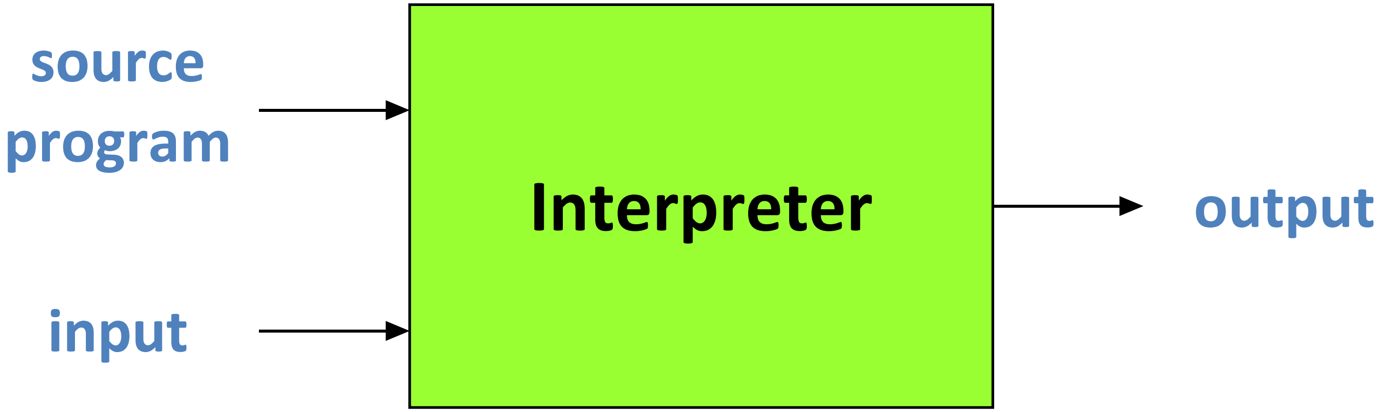
Interesting Past Languages Designed in PLT

- Q-HSK quantum computing language
- Upbeat – sonifying data
- Language to create three-panel comic strips
- Trowel – a language for journalists
- Geometric figure drawing language
- Screenplay animation language
- Manipulation of multiple media
- Escher-like pattern generator
- Functional language for composing music
- What to wear

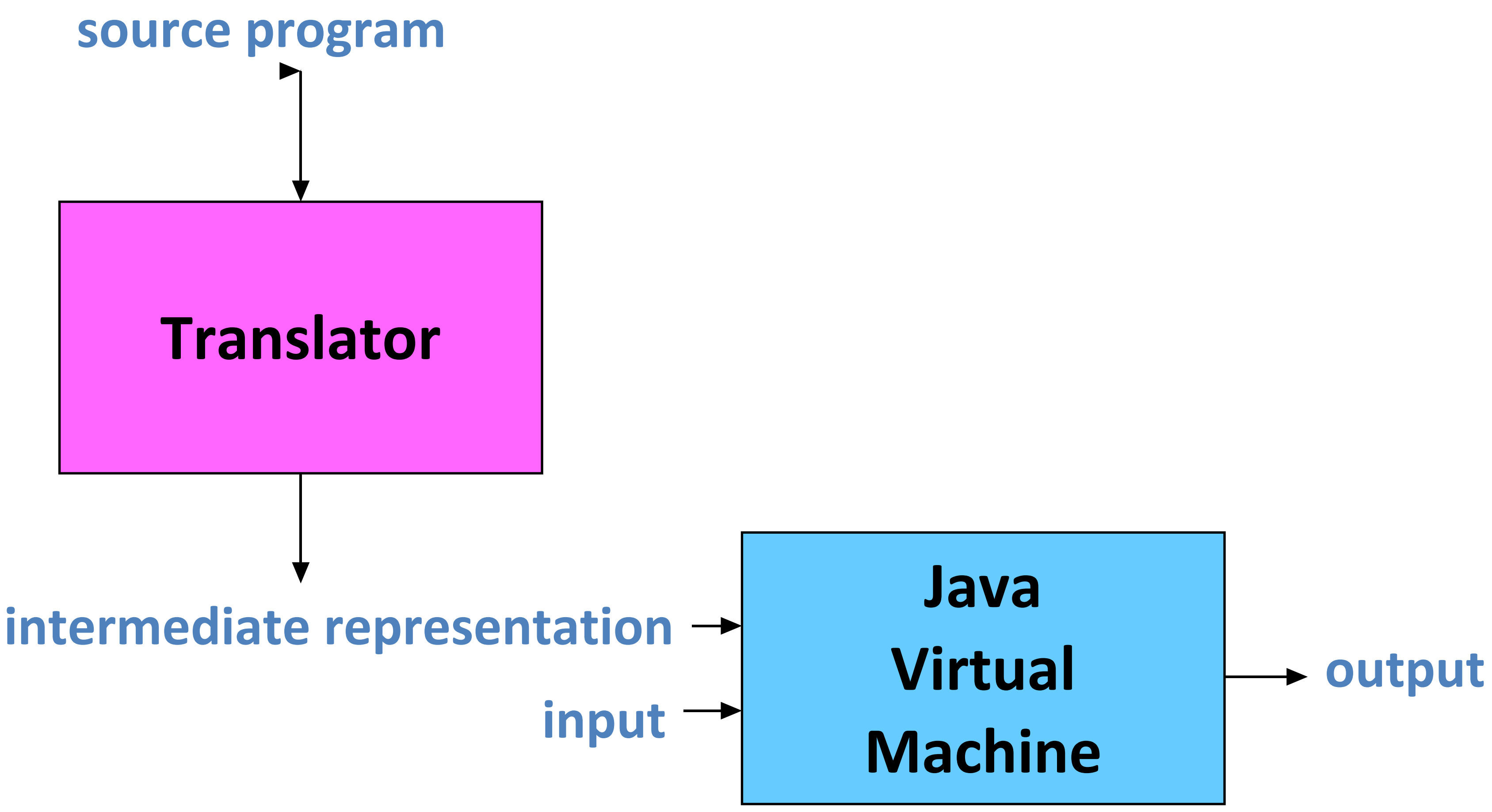
What is a Compiler?



An Interpreter Directly Executes a Source Program on its Input



Java Compiler



Compilers Can Have Many Other Forms

- **Cross compiler:** a compiler on one machine that generates target code for another machine
- **Incremental compiler:** one that can compile a source program in increments
- **Just-in-time compiler:** one that is invoked at runtime to compile each called method in the IR to the native code of the target machine
- **Ahead-of-time compiler:** one that translates IR to native code prior to program execution

Major Application Areas - I

- **Big data**
  - C++, Python, R, SQL, and Hadoop-based languages
- **Scientific computing**
  - Fortran, C++
- **Scripting applications**
  - Awk, Perl, Python, Tcl
- **Specialized applications**
  - LaTex for typesetting
  - SQL for database applications
  - VB macros for spreadsheets

Major Application Areas - II

- Symbolic programming
  - F#, Haskell, Lisp, ML, Ocaml
- Systems programming
  - C, C++, C#, Java, Objective-C
- Web programming
  - CGI
  - HTML
  - JavaScript
  - Ruby on Rails
- Countless other application areas

What does this AWK program do?

```
!x[$0]++
```

Maybe a little less cryptic:

```
!seen[$0]++
```

/* Both programs print the unique lines of the input. */

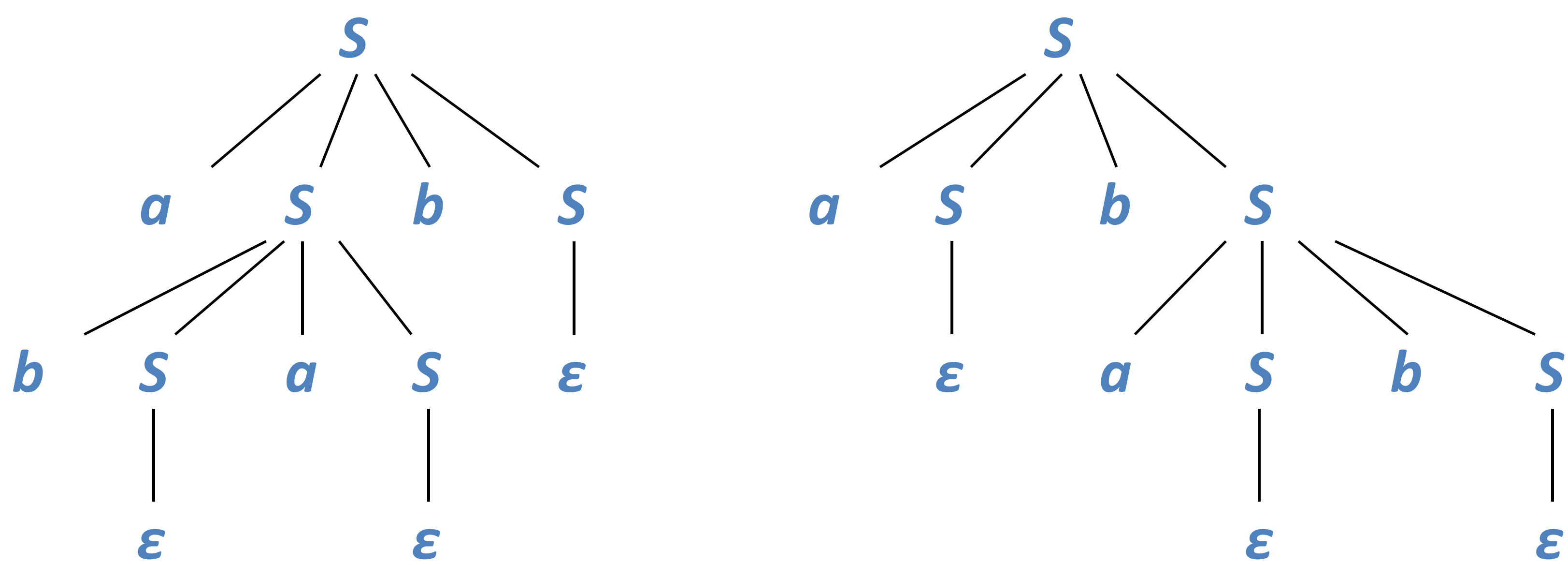
The Specification of PLs

- Syntax
- Semantics
- Pragmatics
- However, a precise, automatable, easy-to-understand, easy-to-implement method for specifying a complete language is still an open research problem

Grammars are Used to Help Specify Syntax

The grammar $S \rightarrow aSbS \mid bSaS \mid \varepsilon$ **generates all strings of *a*'s and *b*'s with the same number of *a*'s as *b*'s.**

This grammar is ambiguous: abab has two parse trees.



$(ab)^n$ has $\frac{1}{n+1}\binom{2n}{n}$ parse trees

Natural Languages are Inherently Ambiguous

I made her duck.

[5 meanings: D. Jurafsky and J. Martin, 2000]

One morning I shot an elephant in my pajamas. How he got into my pajamas I don’t know.

[Groucho Marx, *Animal Crackers*, 1930]

List the sales of the products produced in 1973 with the products produced in 1972.

[455 parses: W. Martin, K. Church, R. Patil, 1987]

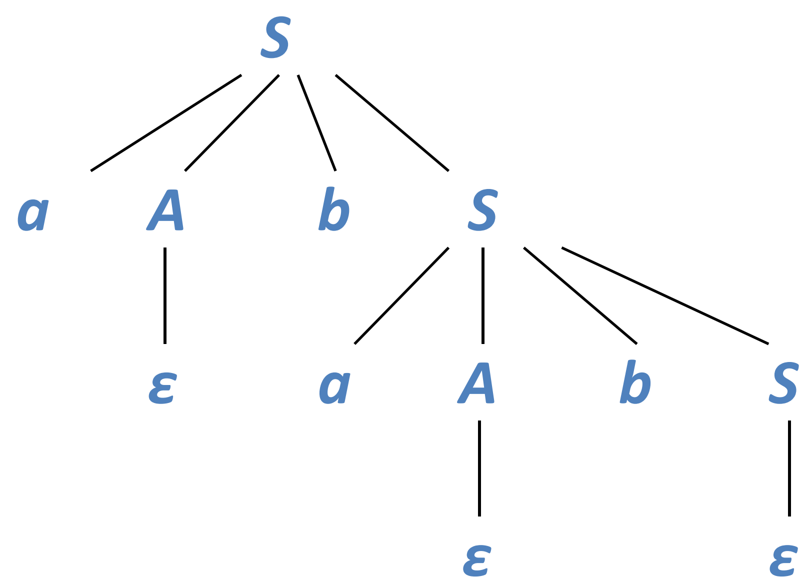
Programming Languages are not Inherently Ambiguous

This grammar *G* generates the same language

$S \rightarrow aAbS \mid bBaS \mid \varepsilon$

$A \rightarrow aAbA \mid \varepsilon$

$B \rightarrow bBaB \mid \varepsilon$



***G* is unambiguous and has only one parse tree for every sentence in *L*(*G*).**

Methods for Specifying the Semantics of Programming Languages

Operational semantics

Program constructs are translated to an understood language.

Axiomatic semantics

Assertions called preconditions and postconditions specify the properties of statements.

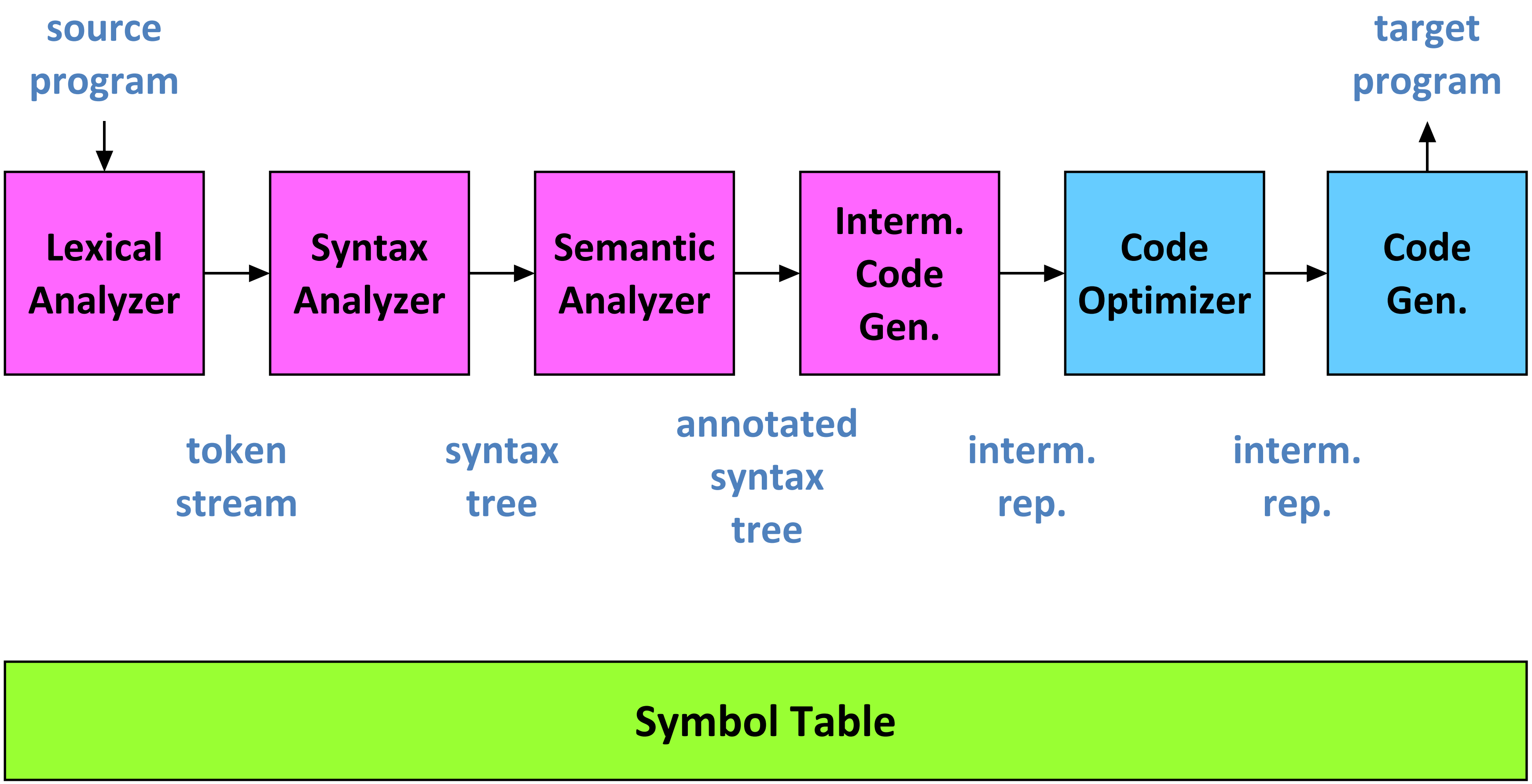
Denotational semantics

Semantic functions map syntactic objects to semantic values.

The Implementation of PLs

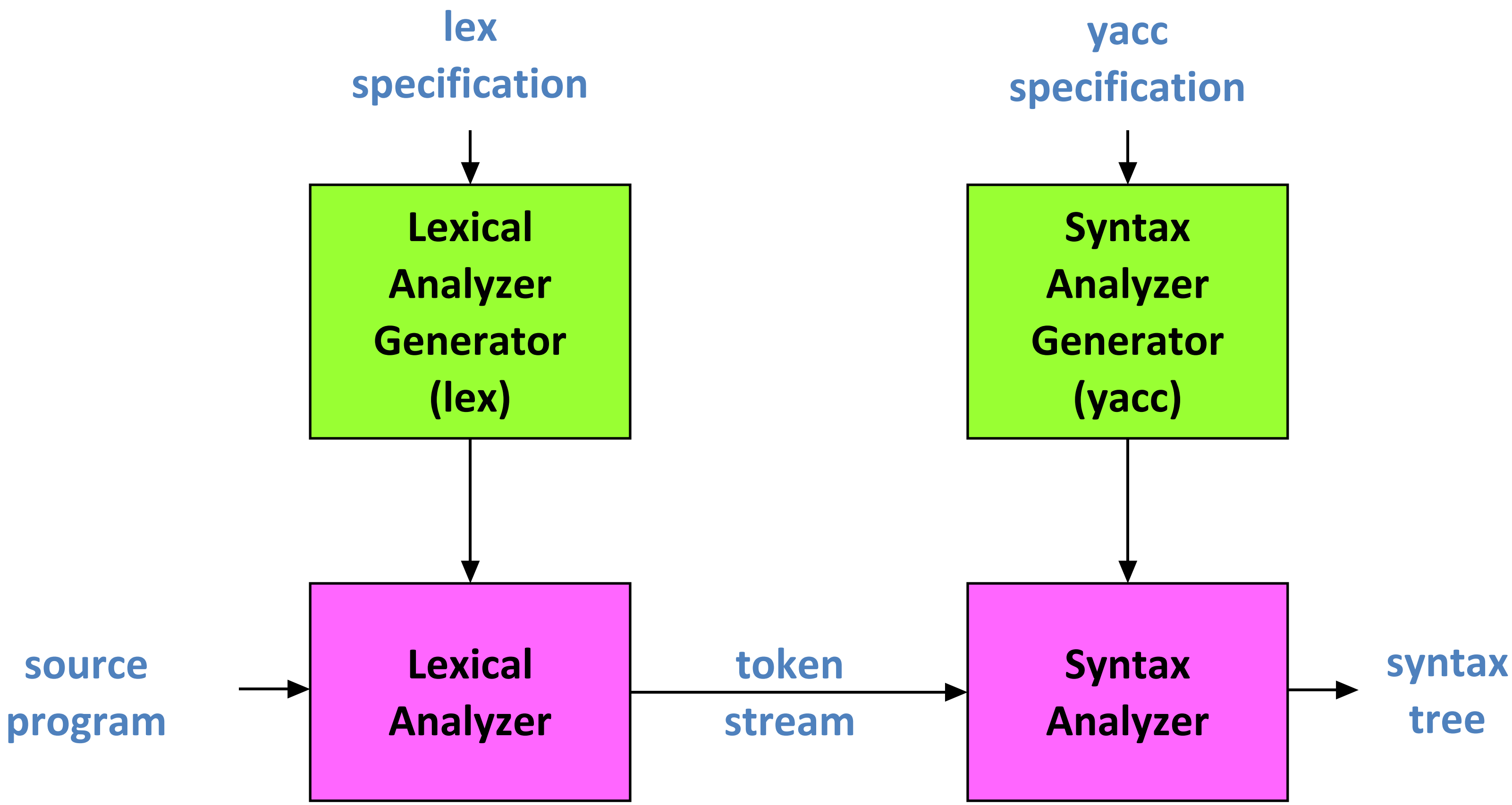
- Compilers
- Interpreters
- Just-in-time compilers
- Compiler collections such as GCC and LLVM

Phases of a Compiler



[A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman, *Compilers: Principles, Techniques, & Tools*, 2007]

Compiler Component Generators



Lex Specification for a Desk Calculator

```
number   [0-9]+\.?|[0-9]*\.[0-9]+
%%
[ ]      { /* skip blanks */ }
{number} { sscanf(yytext, "%lf", &yylval);
           return NUMBER; }
\n|.     { return yytext[0]; }
```

[M. E. Lesk and E. Schmidt, *Lex – A Lexical Analyzer Generator*]

Yacc Specification for a Desk Calculator

```
%token NUMBER
%left '+'
%left '*'
%%
lines : lines expr '\n' { printf("%g\n", $2); }
      | /* empty */
      ;
expr  : expr '+' expr   { $$ = $1 + $3; }
      | expr '*' expr   { $$ = $1 * $3; }
      | '(' expr ')' { $$ = $2; }
      | NUMBER
      ;
%%
#include "lex.yy.c"
```

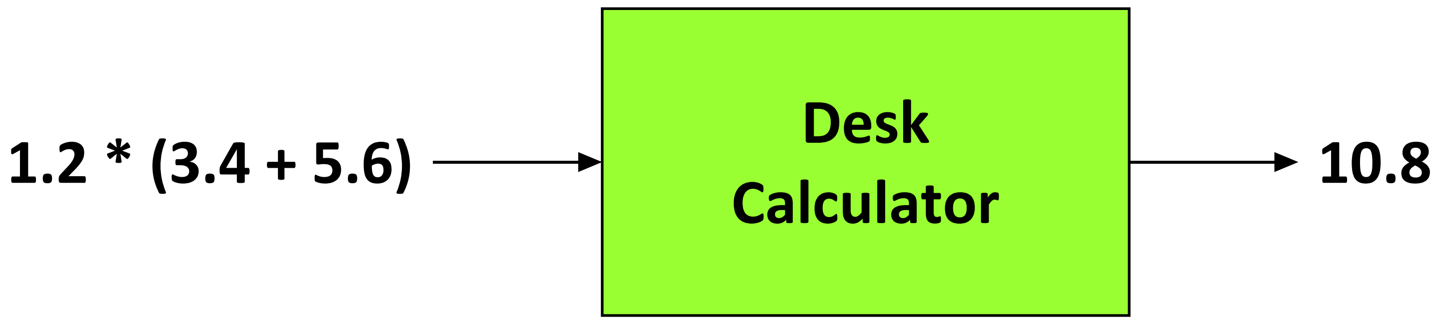
[Stephen C. Johnson, *Yacc: Yet Another Compiler-Compiler*]

Creating the Desk Calculator

Invoke the commands

```
lex desk.l
yacc desk.y
cc y.tab.c -ly -ll
```

Result



The Spring Compilers Course Project

Week	Task
2	Form a **team of five** and design an innovative new language
4	Write a **whitepaper** on your proposed language modeled after the Java whitepaper
8	Write a **tutorial** patterned after Chapter 1 and a **language reference manual** patterned after Appendix A of Kernighan and Ritchie's book, *The C Programming Language*
14	Give a ten-minute **presentation** of the language to the class
15	Give a 30-minute **working demo** of the compiler to the teaching staff
15	Hand in the **final project report**

Team Roles

- **Project manager**
  - sets the project schedule, holds weekly meetings with the entire team, maintains project log, and makes sure project deliverables get done on time
- **Language and tools guru**
  - defines the baseline process to track language changes and maintain the intellectual integrity of the language
  - teaches the team how to use specialized tools used to build the compiler
- **System architect**
  - defines the compiler architecture, modules, and interfaces
- **System integrator**
  - defines system platform, makefile and makes sure components interoperate
- **Tester and validator**
  - defines the test suites
  - "one-click build and test"

Final Project Report

1. **Introduction** – Team
2. **Tutorial** – Team
3. **Reference manual** – Team
4. **Project plan** – Project Manager
5. **Language evolution** – Language Guru
6. **Translator architecture** – System Architect
7. **IDE and runtime environment** – Integrator
8. **Test plan and scripts** – Tester
9. **Lessons learned** – Team
10. **Full code listing** – Team

Telling Lessons Learned in COMS W4115

- **"Designing a language is hard and designing a simple language is extremely hard!"**

- **"During this course we realized how naïve and overambitious we were, and we all gained a newfound respect for the work and good decisions that went into languages like C and Java which we've taken for granted for years."**

Parting Advice

- Grow your language and translator
- Don’t agonize over minor details
- Start immediately!